Object-Oriented Programming Concepts.

1. Abstract Classes…

Abstract classes are when the parent class has a named method, but need its child classes to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

This is the code that explains abstract classes and its child classes;

<?php

// Abstract class definition

abstract class Animal {

    protected $name;

    public function \_\_construct($name) {

        $this->name = $name;

    }

    // Abstract method

    abstract public function makeSound();

    public function describe() {

        return "This is a {$this->name}.";

    }

}

// Subclass implementing abstract method

class Dog extends Animal {

    public function makeSound() {

        return "Bark!";

    }

}

// Subclass implementing abstract method

class Cat extends Animal {

    public function makeSound() {

        return "Meow!";

    }

}

$dog = new Dog("Dog");

echo $dog->describe();

echo $dog->makeSound();

echo $cat ->describe();

echo $cat->makeSound();

?>

1. Interfaces…

It allows you to specify what methods a class should implement without showing how they should be implemented. This is helpful for ensuring the consistency across many classes that may share a behavior.

This is an example showcasing how interfaces can be implemented;

<?php

// Interface definition

interface Animal {

  public function makeSound();

}

// Class definitions

class Cat implements Animal {

  public function makeSound() {

    echo " Meow ";

  }

}

class Dog implements Animal {

  public function makeSound() {

    echo " Bark ";

  }

}

class Mouse implements Animal {

  public function makeSound() {

    echo " Squeak ";

  }

}

class Lion implements Animal {

    public function makeSound() {

      echo " Roar ";

    }

  }

// Create a list of animals

$cat = new Cat();

$dog = new Dog();

$mouse = new Mouse();

$lion = new Lion();

$animals = array($cat, $dog, $mouse, $lion);

// Tell the animals to make a sound

foreach($animals as $animal) {

  $animal->makeSound();

}

?>

1. Traits…

In PHP allow you to reuse methods across multiple classes and they provide a mechanism for code reuse without requiring inheritance, they also can be included in any class and can be combined with class inheritance.

// Define a trait with reusable functionality

trait AnimalTrait {

    protected $name;

    public function setName($name) {

        $this->name = $name;

    }

    public function describe() {

        return "This is a {$this->name}.";

    }

}

// Dog class using the AnimalTrait

class Dog {

    use AnimalTrait;

    public function makeSound() {

        return "Bark!";

    }

}

// Cat class using the AnimalTrait

class Cat {

    use AnimalTrait;

    public function makeSound() {

        return "Meow!";

    }

}

$dog = new Dog();

$dog->setName("Dog");

echo $dog->describe();

echo $dog->makeSound();

echo "<br>";

$cat = new Cat();

$cat->setName("Cat");

echo $cat->describe();

echo $cat->makeSound();

1. Namespaces…

Namespaces are qualifiers that allow better organization by grouping classes that work together to perform a task and they also allow the same name to be used for more than one class.

Namespaces in php are essential in avoiding conflicts between similar class names and functions across different files and libraries.

// Declare namespace for the Controller

namespace MyApp\Controllers;

class AnimalController {

    public function index() {

        return "Animal Controller Index";

    }

}

// Another file with a different namespace

namespace MyApp\Models;

class Animal {

    protected $name;

    public function \_\_construct($name) {

        $this->name = $name;

    }

    public function getName() {

        return $this->name;

    }

}

$controller = new \MyApp\Controllers\AnimalController();

echo $controller->index();

$animal = new \MyApp\Models\Animal("Lion");

echo $animal->getName();

1. Iterables…